

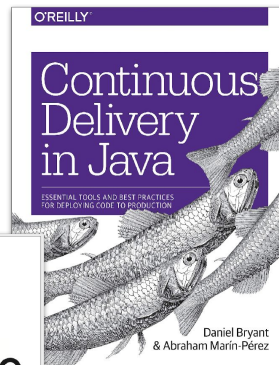
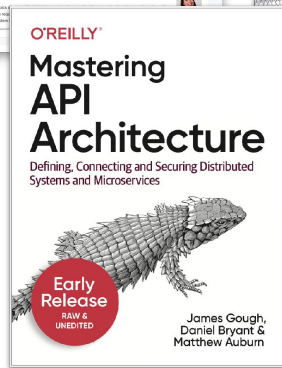
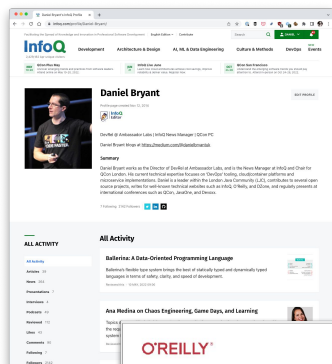
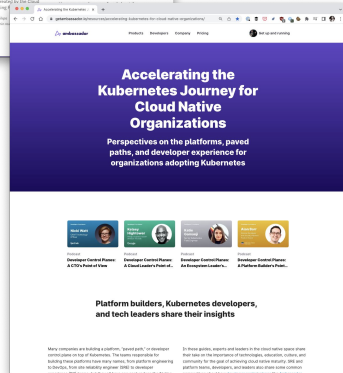
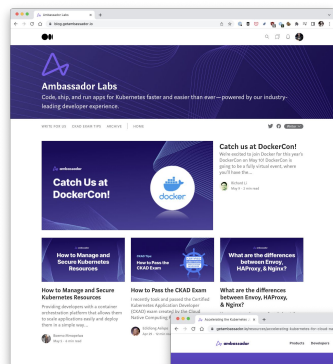
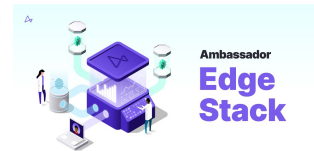
# The Busy Platform Engineers Guide to API Gateways

Daniel Bryant | ~~Head of DevRel, Ambassador Labs~~ Independent Consultant  
@danielbryantuk

# tl;dr

- Choosing (or migrating) an API gateway is a Type 1 decision
- Traffic management is a key part of any platform
- Treat an API Gateway as a product
- Think about developer/operator experience
- Focus on workflows and tooling interoperability

# @danielbryantuk (he/him)



[linktr.ee/danielbryantuk](https://linktr.ee/danielbryantuk)

Decisions, decisions, decisions... 🤔

# Software engineering is all about decisions

"Some decisions are **consequential and irreversible or nearly irreversible** – one-way doors – and these decisions must be made methodically, carefully, slowly, with great deliberation and consultation.

If you walk through and don't like what you see on the other side, you can't get back to where you were before. **We can call these Type 1 decisions.**"

-Jeff Bezos, Founder of Amazon

# Software engineering is all about decisions

"But most decisions aren't like that – they are **changeable, reversible** – they're two-way doors. If you've made a suboptimal **Type 2 decision**, you don't have to live with the consequences for that long. You can reopen the door and go back through.

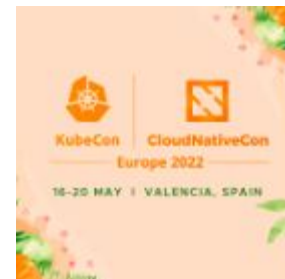
Type 2 decisions can and should be made quickly by high judgment individuals or small groups."

-Jeff Bezos, Founder of Amazon

# Choosing an API gateway is a type 1 decision

- An API gateway is difficult to change/replace
  - “Sticky” technology with a steep learning curve
- On the (business critical) hot path for all traffic
  - All user requests flow through this component
- Can be expensive
  - Contract lock-in is real, yo!
  - Platform engineers need to consider €€€ (especially now)

# Previously at KubeCon '22





# From Kubernetes to PaaS to... err, what's next?

Spoiler alert!

From Kubernetes to PaaS

My answer is **Golden Paths**, a.k.a.

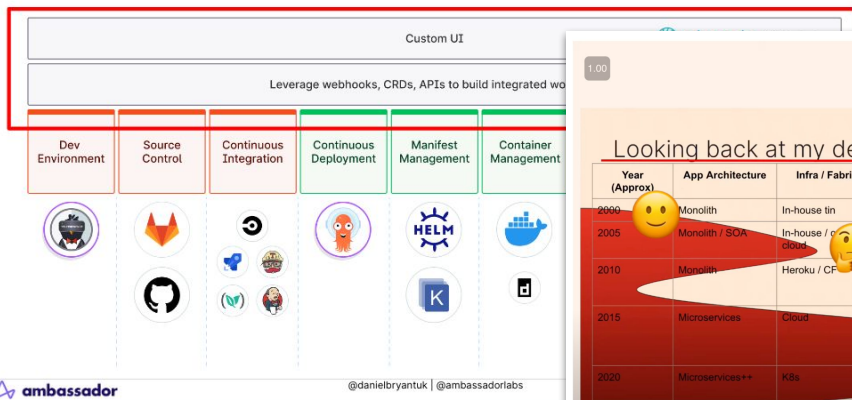
The **real questions** are how many  
and how should **you assemble** them

Platform Engineering

ambassador

@danielbryantuk

The CNCF ecosystem is the foundation for a DCP



Looking back at my dev career Cognitive Load

Year (Approx)	App Architecture	Infra / Fabric	My (Developer) Responsibility	Developer Control Planes
2000	Monolith	In-house tin	Code	IDE, CVS, deploy portal
2005	Monolith / SOA	In-house / cloud	Code, ship, (limited run)	IDE, Mercurial, Jenkins, [ESB, MQs, Gateways]
2010	Monolith	Heroku / CF	Code, run	IDE, Git, Heroku CLI, Heroku UI, New Relic UI
2015	Microservices	Cloud	Code, ship, run	IDE, Git, Docker Hub, Jenkins+plugins, AWS Console, bash, Terraform, Chef...
2020	Microservices++	K8s	Full lifecycle (code, ship, run)++	IDE, Git, K8s

From Kubernetes to PaaS to ... Err, What's Next? - Daniel Bryant, Ambassador Labs

CNCF [Cloud Native Computing Found... 102K subscribers

Subscribed

51

Share

Download

Clip

[youtube.com/watch?v=btUYeOa7JPI](https://youtube.com/watch?v=btUYeOa7JPI)

ambassador

@danielbryantuk | @ambassadorlabs

# A quick recap for building platforms




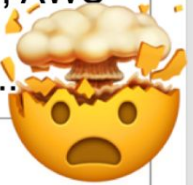
From Kubernetes to PaaS to... err, what's next?

*My answer is **Golden Paths**, a.k.a. paved roads/paths/platforms*

The **real questions** are how much should **you build yourself**,  
and how should **you assemble the control plane** for effective use?

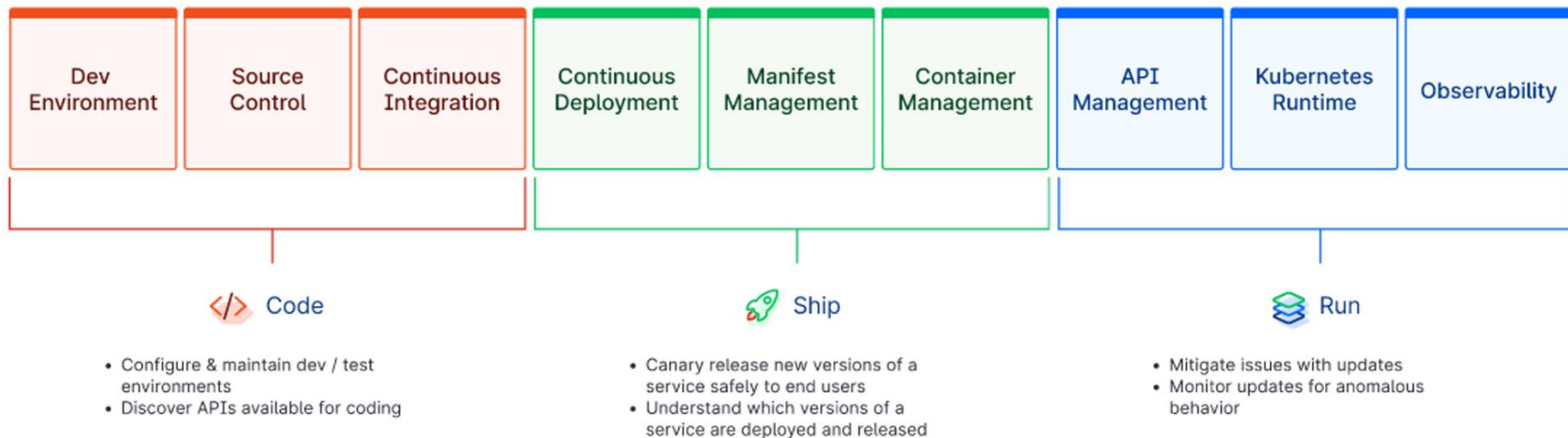
**Platform Engineering** is a how you do this

# Looking back at my dev career *Cognitive Load*

Year (Approx)	App Architecture	Infra / Fabric	My (Developer) Responsibility	Developer Control Planes
2000 	Monolith	In-house tin	Code	IDE, CVS, deploy portal
2005	Monolith / SOA	In-house / cloud 	Code, ship, [limited run]	IDE, Mercurial, Jenkins, [ESB, MQs, Gateways]
2010 	Monolith	Heroku / CF	Code, run	IDE, Git, Heroku CLI, Heroku UI, New Relic UI
2015	Microservices	Cloud	Code, ship, run	IDE, Git, Docker Hub, Jenkins+plugins, AWS Console, bash, Terraform, Chef.. 
2020	Microservices++	K8s	Full lifecycle (code, ship, run)++	IDE, Git, K8s

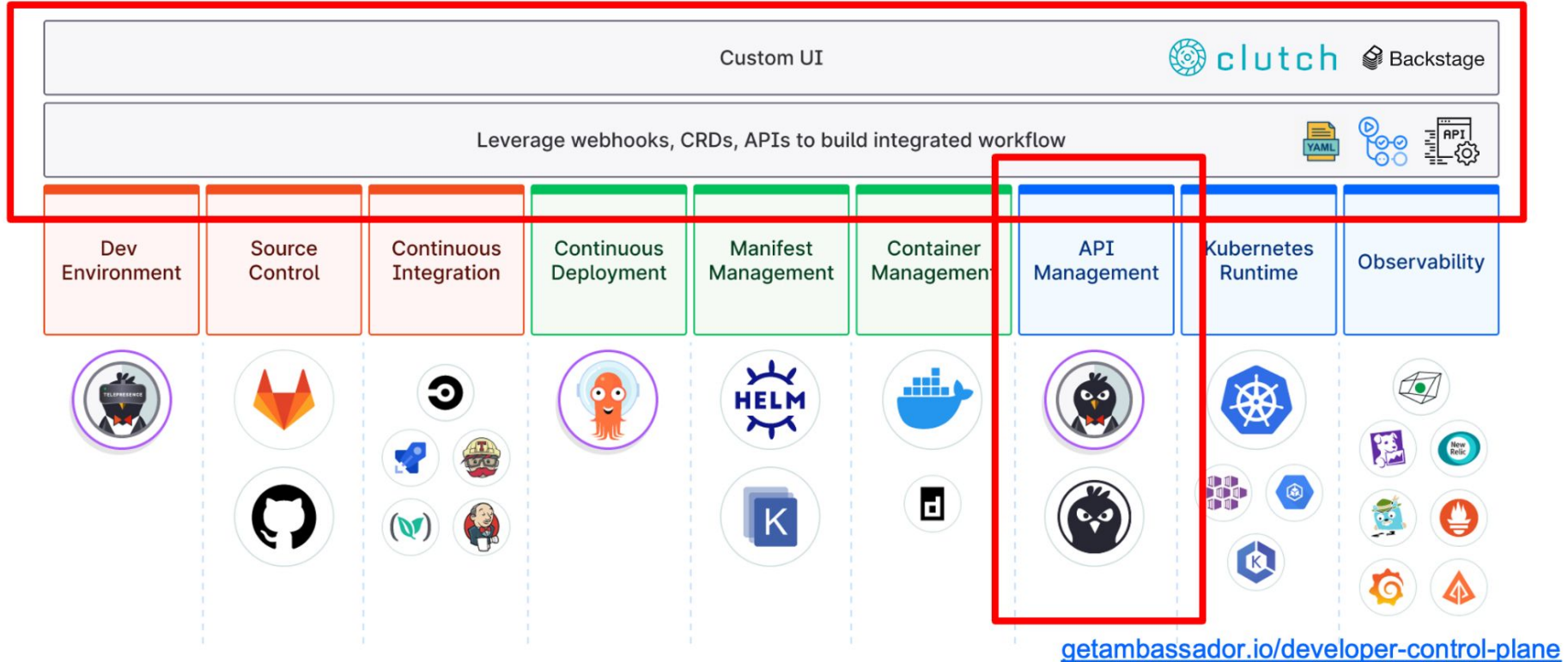
# The need for a platform control plane emerges

?



[getambassador.io/developer-control-plane](https://getambassador.io/developer-control-plane)

# The CNCF ecosystem is the foundation



[getambassador.io/developer-control-plane](https://getambassador.io/developer-control-plane)

# A word of caution with platforms



[twitter.com/tastapod/status/1671810856273707008](https://twitter.com/tastapod/status/1671810856273707008)

# Building platforms: What did I learn?

Treat platform as a product



You can't have good DevX without good UX



Focus on workflows and tooling interoperability



Treat ~~Platform~~ API Gateway as a Product 🚪

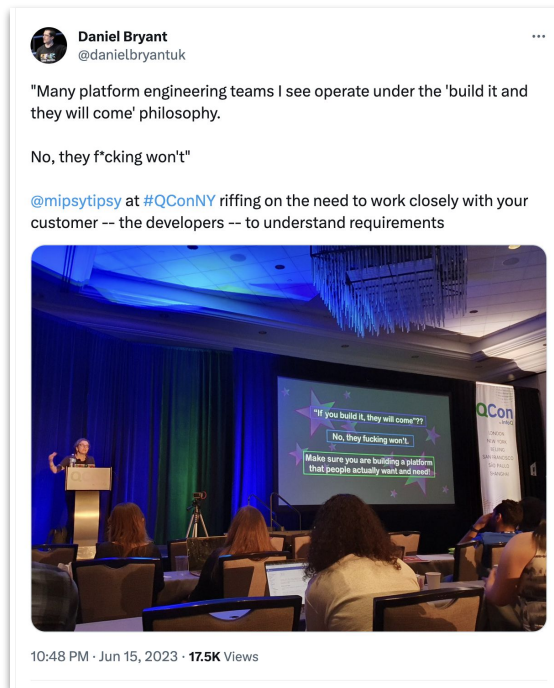


# API Gateway as a Product

- Beware of “product” vs “project”
  - API gateways need a long-term (product) owner
  - Staff and resource an API gateway appropriately
  - If you want build an API gateway, it has to be a product (but don't do this!)
- Take care when lifting and shifting an API Gateway
  - Nearly always end up replatforming (“lift-tinker-and-shift”)

# API Gateway as a Product

- Know your users!
  - API gateways have multiple users (personas)
  - Identify them and their requirements
  - Top down vs bottom up
  - User research is invaluable



[twitter.com/danielbryantuk/status/1669446786354692097](https://twitter.com/danielbryantuk/status/1669446786354692097)

# API Gateway as a Product

- Understand where the API gateway fits into the bigger solution
- The modern cloud native communication stack is complicated
  - The API gateway is only part of the solution



The Past, Present and Future of Cloud Native API Gateways • Daniel Bryant • GOTO 2020

5.2K views • 2 years ago

 GOTO Conferences ✓

Daniel Bryant - Daniel is an expert to the experts with enormous technical insights. Entertaining and Inspiring ABSTRACT Many ...

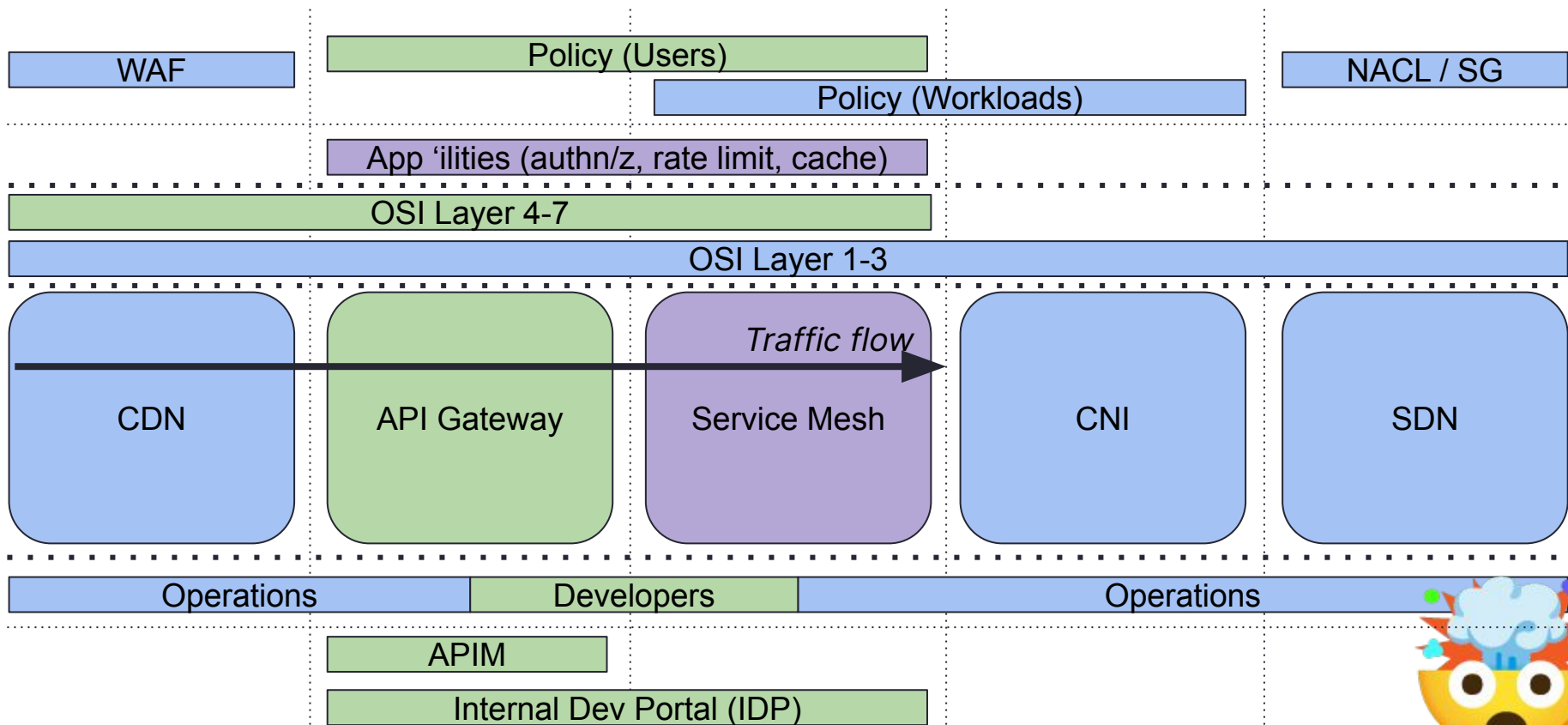


Intro | tldr | Edge: The boundary between your data center & your user(s) | Thesis | History | ...

10 chapters ▾

<https://www.youtube.com/watch?v=Q09dAnIN4RY>

# Modern cloud native comms stack



# All-in-one or one-for-all?

- You can implement “all-in-one” solutions
  - Solo
  - Isovalent
  - Kong
  - Cloud vendors(?)
- Or choose best of breed for each component
  - Ambassador's Emissary-ingress, Envoy Gateway
  - Buoyant's Linkerd, HashiCorp's Consul
  - Cloud vendor CNI
  - Cloudflare



You can't have good DevX  
without good UX 

# You can't have good DevX without good UX

- Understand the approach and defaults for your platform
  - Kubernetes native (CRDs, GitOps-friendly)
  - CLI or API-driven
  - UI-driven
- Tailor the experience to personas
  - Developer experience
  - Operator experience
- Platform engineering tenet: self-service
  - But this means many things to many people
  - PRs vs biz-focused clickops



<https://twitter.com/danielbryantuk/status/1557268926429528065>



# Self-Service Configuration

```
apiVersion: getambassador.io/v3alpha1
kind: Mapping
metadata:
```

```
  apiVersion: getambassador.io/v3alpha1
  kind: Mapping
  metadata:
```

```
    name: restricted-mapping
  spec:
    kind: Mapping
    metadata:
      name: restricted-mapping
    spec:
      host: restricted.example.com
      prefix: /restricted/
      rewrite: /a/very/safe/path/
      rewrite_host: safe.example.com
      service: dangerous-service
```

```
apiVersion: getambassador.io/v3alpha1
kind: Listener
metadata:
```

```
  apiVersion: getambassador.io/v3alpha1
  kind: Host
  metadata:
```

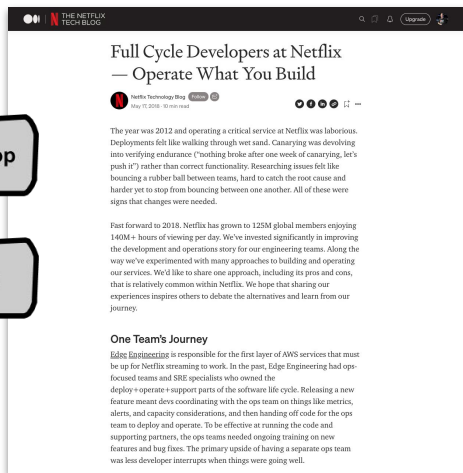
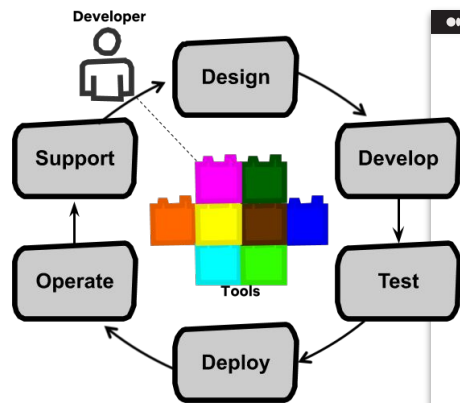
```
    apiVersion: getambassador.io/v3alpha1
    kind: AuthService
    metadata:
      name: extauth-service
    spec:
      auth_service: example-auth
      path_prefix: "/extauth"
      allowed_request_headers:
        - "x-example-session"
      allowed_authorization_headers:
        - "x-example-session"
        - "x-example-userid"
```



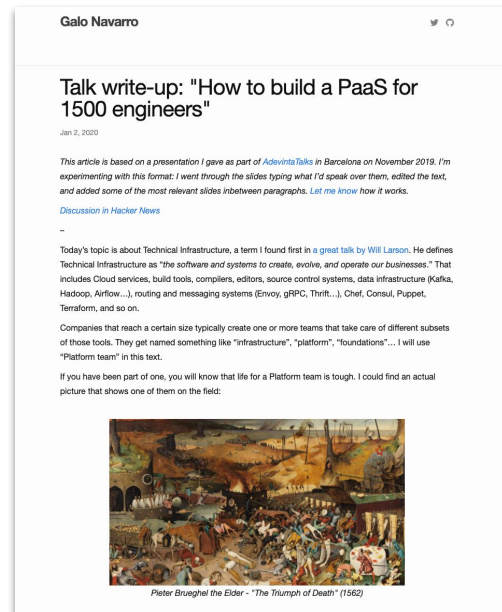
Focus on workflows and  
tooling interoperability 🏭🤝

# Workflow and interoper

“[The] centralized [platform] teams act as force multipliers by turning their specialized knowledge into reusable building blocks.”



[netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249](https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249)



“A good deal of the job is ultimately about finding the right balances between standardization and autonomy”

[svraroa.github.io/paas/infrastructure/platform/kubernetes/cloud/2020/01/02/talk-how-to-build-a-paas-for-1500-engineers.html](https://svraroa.github.io/paas/infrastructure/platform/kubernetes/cloud/2020/01/02/talk-how-to-build-a-paas-for-1500-engineers.html)



# Self-Service Configuration

```
apiVersion: getambassador.io/v3alpha1
kind: Mapping
metadata:
```

```
  apiVersion: getambassador.io/v3alpha1
  kind: Mapping
  metadata:
```

```
    name: restricted-mapping
  spec:
    kind: Mapping
    metadata:
      name: restricted-mapping
    spec:
      host: restricted.example.com
      prefix: /restricted/
      rewrite: /a/very/safe/path/
      rewrite_host: safe.example.com
      service: dangerous-service
```

```
apiVersion: getambassador.io/v3alpha1
kind: Listener
metadata:
```

```
  apiVersion: getambassador.io/v3alpha1
  kind: Host
  metadata:
```

```
    apiVersion: getambassador.io/v3alpha1
    kind: AuthService
    metadata:
      name: extauth-service
    spec:
      auth_service: example-auth
      path_prefix: "/extauth"
      allowed_request_headers:
        - "x-example-session"
      allowed_authorization_headers:
        - "x-example-session"
        - "x-example-userid"
```



# Separation of Concerns

```
apiVersion: getambassador.io/v3alpha1
kind: Mapping
metadata:
```

```
apiVersion: getambassador.io/v3alpha1
spec:
  kind: Mapping
  metadata:
```

```
apiVersion: getambassador.io/v3alpha1
spec:
  kind: Mapping
  metadata:
    name: restricted-mapping
  spec:
    host: restricted.example.com
    prefix: /restricted/
    rewrite: /a/very/safe/path/
    rewrite_host: safe.example.com
    service: dangerous-service
```

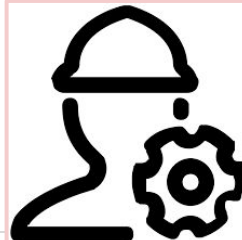


ambassador

```
apiVersion: getambassador.io/v3alpha1
kind: Listener
metadata:
```

```
apiVersion: getambassador.io/v3alpha1
spec:
  kind: Host
  metadata:
```

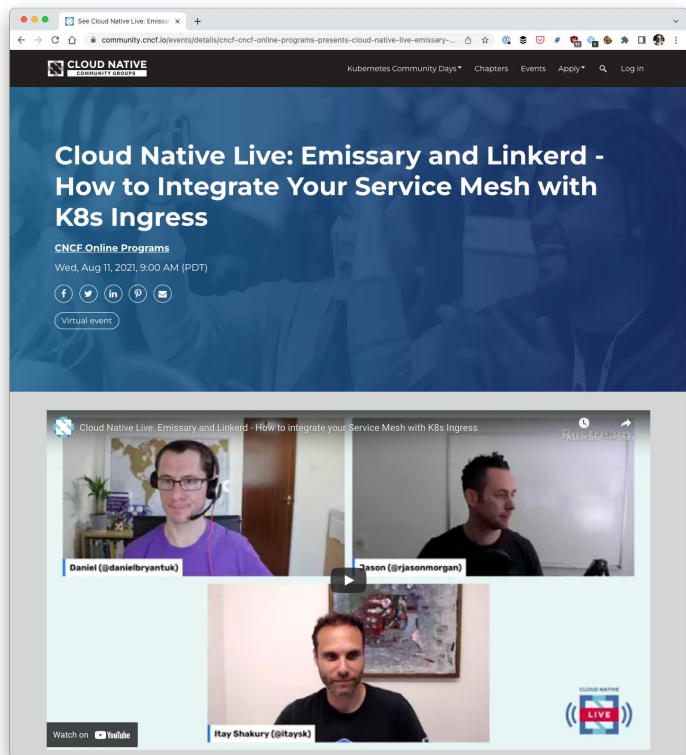
```
apiVersion: getambassador.io/v3alpha1
spec:
  kind: AuthService
  metadata:
    name: extauth-service
  spec:
    auth_service: example-auth
    path_prefix: "/extauth"
    allowed_request_headers:
      - "x-example-session"
    allowed_authorization_headers:
      - "x-example-session"
      - "x-example-userid"
```



# Extra validation when applying global configuration?

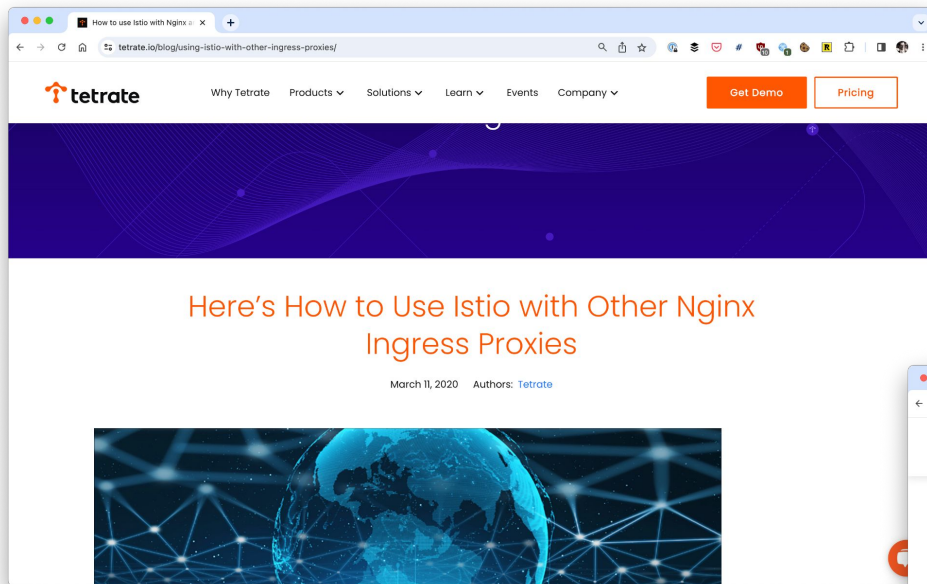


# Interop Example: Emissary-ingress & Linkerd

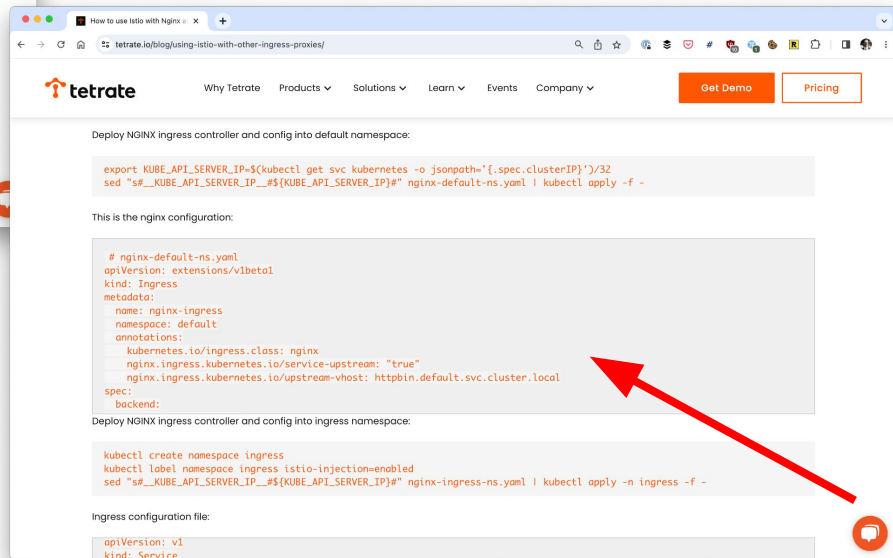


- CNCF projects
  - Emissary-ingress: n/s gateway
  - Linkerd: e/w service mesh
- Both use [Kubernetes Resource Model \(KRM\)](#)
  - CRDs, controllers, best practices
- One line integration

```
kubectl -n emissary get deploy emissary-ingress -o yaml | \
linkerd inject --skip-inbound-ports 80,443 - | \
kubectl apply -f -
```
- Similar configuration across projects



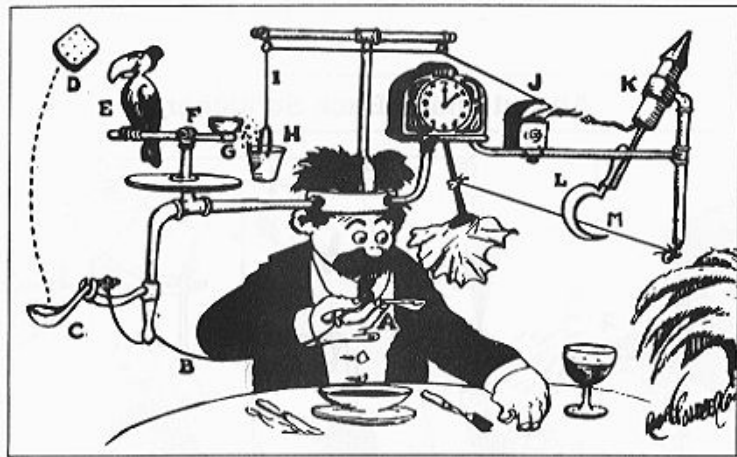
[tetrade.io/blog/using-istio-with-other-ingress-proxies/](https://tetrade.io/blog/using-istio-with-other-ingress-proxies/)



# API gateway plugins: love 'em/hate 'em

- Plugins/extension/filters provide:
  - Reusability
  - Separation of concerns
  - Performance (?)
- But, they are often highly coupled
  - With the API gateway
  - With the system itself
- Please, please, please don't put business logic in them!
  - I've seen this way too many times

Real-life API gateway plugin architecture





# Wrapping up 🎉

# Tell me more about my (K8s) API Gateway options

Products/Project	Ingress Nginx	Kong	Apache APISIX	Azure App Gateway	Nginx+	HAProxy Tech	HAProxy (Community)	Voyager	Istio Ingress	Contour	Ambassador-ingress	Gloo	Traefik	Skipper	Citrix Ingress	GKE Ingress	ALB Ingress	AKO	KrakenD	Tyk
1. General info																				
2. Protocols																				
3. Clients																				
4. Traffic routing																				
5. Upstream resiliency																				
6. Load balancer strategies																				

LearnK8s: <https://docs.google.com/spreadsheets/d/191WWNpjJ2za6-nbG4ZoUMXMpUK8KIClosvQB0f-oq3k/edit?usp=sharing>

# Conclusion

- Choosing (or migrating) an API Gateway is a Type 1 decision
  - Tricky to reverse: but the right decision adds a lot of value
  - Clear ownership needed for platform and API gateway
- Treat API Gateways as a product
  - Identify personas and requirements
  - Integration within the wider cloud native comms stack is key
- Think about developer/operator experience
  - Self-service for the win!
- Focus on workflows and tooling interoperability
  - Good integration and appropriate extensions are the key!

# Thank you!

@[danielbryantuk](#)

[Improving Cloud Native DevEx: The API Gateway Perspective](#)



[Moving to the Cloud: Exploring the API Gateway to Success](#)



[Platform Engineering Guide](#)

